MMBench: Benchmarking End-to-End Multi-modal DNNs and Understanding Their Hardware-Software Implications

Cheng Xu[1] Xiaofeng Hou[1] Jiacheng Liu[1] Chao Li[1] Tianhao Huang[1] Xiaozhi Zhu[1]
Mo Niu[1] Lingyu Sun[1] Peng Tang[1] Tongqiao Xu[1]
Kwang-Ting Cheng[2] Minyi Guo[1]

[1]*Shanghai Jiao Tong University*
[2]*Hongkong University of Science and Technology*

Abstract

The explosive growth of various types of big data and advances in AI technologies have catalyzed a new type of workloads called multi-modal DNNs. Multi-modal DNNs are capable of interpreting and reasoning about information from multiple modalities, making them more applicable to real-world AI scenarios. In recent research, multi-modal DNNs have outperformed the best uni-modal DNN in a wide range of distributed computing applications from traditional multimedia systems to emerging autonomous edge systems. However, despite their importance and superiority, very limited research attention has been devoted to understand the characteristics of multi-modal DNNs and their implications on current computing software/hardware platforms. Existing benchmarks either target uni-modal DNNs or only focus on the algorithm characteristics of multi-modal DNNs. There lacks representative benchmark suites that provide comprehensive system and architecture level analysis of multi-modal networks.

To advance the understanding of these multi-modal DNN workloads and facilitate related research, we present MMBench, an open-source, end-to-end benchmark suite consisting of a set of real-world multi-modal DNN workloads with relevant performance metrics for evaluation. We then use MMBench to conduct an in-depth analysis on the characteristics of multi-modal DNNs. We demonstrate their unique characteristics of clear multi-stage execution, frequent synchronization and high heterogeneity, which distinguish them from conventional uni-modal DNNs. Finally, we conduct a case study and extend our benchmark to edge devices. We hope that our work can provide insights for future software/hardware design and optimization to underpin multi-modal DNNs on both cloud and edge computing platforms.

1. Introduction

Multi-modal deep neural networks (DNNs) have attracted significant attention [7, 43, 53] in recent years. By fusing information from a variety of modalities, they can provide higher prediction accuracy than the best traditional uni-modal DNNs [19, 24, 53]. In fact, multi-modal DNNs have been shown to outperform the best uni-modal DNNs by 5% - 30% accuracy in many important application fields [32]. Furthermore, the development of perception technology and AI accelerators has facilitated the deployment and development of multi-modal DNNs in a wide range of real-world applications from conventional multimedia to emerging autonomous systems.

Despite their superiority in performance, multi-modal DNNs possess several unique characteristics, that have never been explored before and would pose new challenges to system and architecture designs previously applied to uni-modal DNNs. These characteristics include:

- **Three-stage Execution Pattern:** Most multi-modal DNN applications follow a common three-stage execution pattern. In the first stage, known as *encoder*, independent neural networks are utilized to translate input modalities to distinct representations that are suitable for machine learning. These representations are then fed to the second stage, known as *fusion* where they are federated. Finally, the task-specific head network produces the final results in the third stage, known as *head*. The three stages are executed in serial, and each stage exhibits different execution and resource usage patterns.
- **Intra-network Heterogeneity:** A multi-modal DNN shows great intra-network heterogeneity due to the use of different encoder and fusion networks. The first stage inherently involves different networks and operators to process different modalities e.g. CNNs for image modality and RNN/Transformers for text modality. Additionally, in the fusion stage, different fusion methods can be applied to federate the features of different modalities for different accuracy targets. As a result, there are no universal architectural solutions to optimize all modalities and stages, which are often dominated by heterogeneous operations.
- **Frequent Synchronization:** The fusion stage in multi-modal DNNs incurs substantial synchronization operations compared to traditional uni-modal networks. In this stage, the fusion network waits for the completion of all modalities, and additional CPU-GPU synchronization is needed to process in-

termediate data, such as the feature maps generated from various modalities. These frequent synchronization operations can become a key performance bottleneck for multi-modal DNN computation, as they add extra latency and overhead.

As multi-modal DNNs become increasingly popular and differ significantly from conventional uni-modal DNNs, it is crucial to understand these unique characteristics and their implications for system and architecture designs. It is preferable to analyze their features and assign agile management strategies to maximize overall efficiency [15]. However, there is currently a lack of a well-designed benchmark suite that provides system- and architecture-level characterization of multi-modal DNNs. On one hand, uni-modal DNN benchmarks in previous studies [13, 14, 36] cannot be directly applied to multi-modal DNNs due to the differences in their characteristics. On the other hand, existing multi-modal DNN benchmarks [24] only focus on algorithm-level features such as accuracy, model complexity and robustness without providing any analysis of system and architecture. Therefore, there is a strong motivation to develop benchmarks/tools specialized for multi-modal DNN applications and to explore their implications on today's computing architecture and systems.

In this paper, we propose MMBench, an end-to-end benchmark suite for multi-modal DNN applications. MMBench covers a wide spectrum of representative multi-modal applications across multiple major research areas. We also leverage MMBench to study the characteristics of multi-modal DNNs and their implications across the execution stacks. To the best of our knowledge, MMBench is the first benchmark suite specialized in architecture and system research in multi-modal computing. We design MMBench with the following principles:

- **Representativeness.** We construct MMBench using 9 end-to-end multi-modal DNN workloads from five of the most representative application domains, which cover traditional applications like multimedia and emerging domains such as autonomous driving. This approach ensures that MMBench is representative of multi-modal applications in use today.
- **Thoroughness.** We ensure that properties such as modality types, fusion methods, network structures in MMBench are diverse and cover a wide range of multi-modal DNNs in different domains. This level of thoroughness provides researchers with a detailed understanding of performance and potential areas for improvement for multi-modal DNNs.
- **Comprehensiveness.** At MMBench , we have gone beyond offering just operational workloads and result scoreboards. We also provide comprehensive profiling tools and insights at the architecture and system levels. This level of support enables researchers to build on our work and advance the state of the art in multi-modal computing.

The rest of the paper is organized as follows. First, Section 2 provides the background and related work. Section 3 details the designs of MMBench. Section 4 shows the experimental methodologies and highlights the key features of multi-modal DNNs and their hardware-software implications. Section 5 gives two case studies that demonstrate how MMBench guides the system and architecture designs. Finally, Section 6 concludes this paper.

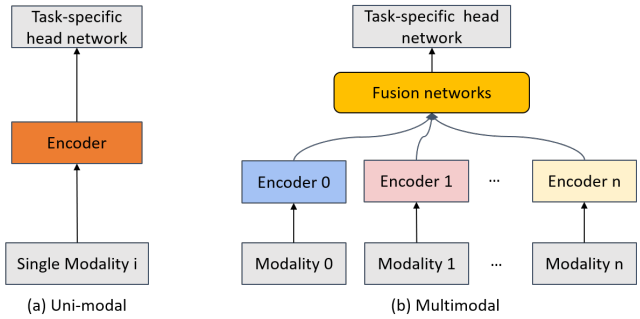


Figure 1: Schematic diagram of multi-modal and uni-modal network structures.

TABLE 1: Commonly used fusion operators [50, 51].

Fusion type	**Formulation of $F(x, y)$**	**Meaning**
Zero	0	Discards these features
Sum	$x + y$	Sum features
Concat	$\text{ReLU}(\text{Concat}(x, y)W + b)$	Concat features
Tensor	$x \otimes y$	Outer product-based attention
Attention	$\text{Softmax}(\frac{xy^T}{\sqrt{C}}y)$	Use attention mechanism
LinearGLU	$\text{GLU}(xW_1, yW_2)$ $= xW_1 \odot \text{Sigmoid}(yW_2)$	linear layer with the GLU

2. Background and Related Work

2.1. Basics of Multi-modal DNNs

Multi-modal DNN is a kind of neural network that learns and improves through the experience of data from multiple modalities. Figure 1 shows the common structure of a multi-modal DNN compared to a uni-modal DNN. At a higher level, the multi-modal DNNs fuse the features from multiple modalities to produce more accurate predictions. Specifically, it consists of three main stages. In the first stage, input modalities are transferred to distinct representations suitable for machine learning by various representation learning methods such as CNNs. In the second stage, it leverages a fusion model to generate the multi-modal representation by federating these processed uni-modal representations. Finally, the multi-modal representation is fed into the task-specific network to produce the final prediction.

Multi-modal DNNs have been demonstrated to outperform the uni-modal ones in various application fields [18, 24, 39]. Most of the current studies employ pretrained DNN backbone models as modality encoders and mainly focus on finding more effective fusion or representation methods of different modalities [43, 52, 53]. Commonly adopted fusion operators are presented in Table 1. Besides, the fusion technique can further be categorized into two main classes, namely early fusion methods [57] and late

fusion methods [6, 45] depending on the depth of encoders to execute before fusion. Among these methods, Zhou et al [57] used a multiple discriminant analysis scheme to implement an early fusion approach that concatenates different modality features. Uperkernel learning [45] is the representative method of late fusion. Recently, motivated by the ability of transformers [41], a branch of works use multi-modal transformers to model different modalities [49].

With the significant performance advance, mutli-modal DNNs have been widely adopted in various scenarios. Thus it is in urgent need of benchmarking multi-modal DNNs from system and architectural level to facilitate the optimization in their deployment.

2.2. Benchmarking Conventional DNNs

Previous researches have paid extensive attention to characterizing features of uni-modal DNN applications [20, 44, 46, 47]. We can broadly classify these works into three types according to their associated evaluation metrics: algorithm-oriented DNN benchmarks, architecture-oriented DNN benchmarks and DNN simulation frameworks. Among them, algorithm-oriented DNN benchmarks [3, 10, 36, 37] strive to incorporate and build a collection of representative DNN models to empower the performance and accuracy comparison of different DNN training and inference. Architecture-oriented DNN benchmarks [5, 54, 55] target on analyzing the architectural features of DNNs on computing systems of different sizes. MLPerf [36] is a comprehensive benchmark for measuring ML inference performance across a spectrum of use cases. MDLBench [55], Embench [5] and AIoTBench [26] are representative benchmarks that characterize the features of different AI models on edge or mobile devices while NNBench-X [48], GNNMark [8] target on acceleration hardware design for different DNNs.

However, the system and architecture level implications drawn by uni-modal DNN benchmarks can not be directly applied to multi-modal DNNs. Compared with uni-modal DNNs, multi-modal DNNs possess several unique characteristics such as clear stage divisions, frequent synchronization and high workload heterogeneity [19]. There lack of specialized architecture-oriented multi-modal benchmarks.

2.3. Benchmarking Multi-modal DNNs

Some efforts have also been made in benchmarking the emerging workload of multi-modal DNNs. MultiBench [24] is a well-known benchmark suite in multi-modal algorithm research. MultiBench implements a wide spectrum of multi-modal applications and provides a reliable way to evaluate the performance across domains and modalities, the complexity during training/inference and the robustness to noisy and missing modalities. However, it only evaluates multi-modal DNNs from the algorithm aspect and does not provide system and architecture-level insights. Besides, MultiBench does not provide end-to-end implementation, which makes it insufficient to support architecture research especially for edge devices where raw data are collected from sensors and processed locally.

TABLE 2: Comparison of MMBench and other benchmarks [10, 24, 36, 40]. *H* refers to hardware, *Ar* refers to architecture, *S* refers to system, *Al* refers to algorithm.

Benchmarks	Uni-modal DNN			Multi-modal DNN	
	MLPerf	DAWNBench	AIBench	MultiBench	Ours
Applications	5	3	10	15	9
Objectives	H	H/Ar	H	Al	H/Ar, S/Al
Cloud	✓	✓	✓	✓	✓
Edge	✓	x	x	x	✓
End-to-End	x	✓	✓	x	✓
Easy-to-Use	x	x	x	x	✓

Architecture benchmarks such as MLPerf [36], DAWNBench [10], and AI-Benchmark [40] can be applied to benchmarking multi-modal DNNs but requires significant modifications. MLPerf provides a comprehensive analysis of how fast systems can train and inference to a target quality metric while covering a wide range of models and areas. It measures a wide range of metrics such as training time, training cost, inference latency, and inference cost. However, as general-purpose full-system benchmarks, they lack specialized algorithm awareness and corresponding analysis in this emerging area. Specialized multi-modal benchmarks can better help design and deploy efficient multi-modal DNN systems.

2.4. The Missing Piece of Multi-modal Research

Multi-modal DNNs are applied in a wide range of applications in different fields. Generally, multi-modal DNNs are more computing-intensive compared with traditional uni-modal DNNs, which may possibly lead to the problem of QoS and power budget violation [16, 17]. In data centers, we need to analyze complex multi-modal data for the highest algorithm performance (e.g., image- and text-based intelligence applications); in edge devices, we need to process raw data collected by multiple sensors locally with limited computational resources within QoS (e.g., autonomous driving). Supporting the inference of such diverse and heterogeneous workloads with high energy efficiency and low latency is becoming a great challenge.

In order to support efficient reasoning on multi-modal networks in data centers and edge devices, there is an urgent need for benchmarks that can accurately model the system and architecture level characteristics of multi-modal networks. However, there have been no such well-designed benchmark suites as presented in table 2. On one hand, the architecture-oriented DNN benchmarks have not covered this emerging research area yet. The implications of uni-modal DNNs can not be directly applied to multi-modal DNNs. On the other hand, existing multi-modal benchmarks all focus on analyzing algorithm-level characteristics. Thus, we present MMBench in this paper to bridge this gap and benefit further research in this area.

3. The MMBench Suite

In this section, we introduce the unique features of MMBench and the specific benchmark setup, i.e. the workloads and the profiling pipeline.

TABLE 3: Characteristics of each applications in MMBench

Application domain	Multimedia Application		Affective Computing		Intelligent Medicine		Smart Robotics		Automatic Driving
Workload	*AV-mnist*	*MM-imdb*	*CMU-mosei*	*MUStARD*	*Medical VQA*	*Medical Seg.*	*Mujoco Push*	*Vision & Touch*	*TransFuser*
Model size	Small	Large	Large	Large	Large	Medium	Medium	Medium	Medium
Modalties	*1.image, 2.audio*	*1.image, 2.text*	*1.language, 2.vision, 3.audio*	*1.language, 2.vision, 3.audio*	*1.image, 2.text*	*MRI scans (T1, T1c, T2, Flair)*	*1.position, 2.sensor, 3.image, 4.control*	*1.image, 2.force, 3.proprioception, 4.depth*	*1.image 2.LiDAR*
Encoders	*1,2: LeNet*	*1: VGG 2: Albert*	*1: BERT 2: OpenFace 3: Librosa*	*1: BERT 2: OpenFace 3: Librosa*	*1: DenseNet 2: Roberta*	*All: U-Net*	*1,2,4: MLP 3: CNN*	*1,2,4: CNN 3: MLP*	*1,2: ResNet*
Fusion methods	*Concate, tensor*	*Concate, tensor*	*Concate, tensor, transformer*	*Concate, tensor, transformer*	*Transformer*	*Transformer*	*Concate, tensor, transformer*	*Concate, tensor*	*transformer*
Task	Class.	Class.	Reg.	Class.	Gen.	Seg.	Class.	Class.	Class.

3.1. Key Features of MMBench

Besides the general design principles, MMBench possess the following unique features closely related with the characteristics of multi-modal DNNs, which distinguishes it from general-purpose benchmarks in this specific area:

- **Fine-grained Network Characterization.** From network structure level, multi-modal DNNs can be viewed as the assembly of multiple encoder networks, fusion network and head network, which require more fine-grained workload characterization [24, 51]. It is inaccurate to use the average or max value of the entire application to characterize multi-modal DNNs since these sub-nets may greatly differ in execution pattern and resource usage. MMBench provides options to split the multi-modal DNN into different stages and characterize the sub-nets respectively.
- **End-to-End Application Execution.** From application level, the processing of raw data for multi-modal DNNs is time-consuming and often require end-to-end execution in real-world scenarios [25, 35]. Many networks take processed data as input [9, 42]. Existing algorithm-oriented benchmarks tend to ignore these preprocessing and provide links to the processed data [24]. Ignoring the preprocessing part results in bias on the computing process. MMBench provides an end-to-end multi-modal processing benchmark that can help us understand the full computational process of multi-modal networks.
- **User-friendly Profiler Integration.** From the architecture profiler level, it is often unnecessary and time-consuming to utilize the entire dataset. A dataset-free computation abstraction can significantly ease the profiler usage. Many datasets in multi-modal neural network research are not open-sourced and require a lengthy application process. Besides, some datasets can take up to hundreds of Gigabyte [35]. MMBench still provides models and links to the datasets to prove that all applications are of high performance. However, MMBench also provides the option to abstract the computation when network accuracy is not needed. It can randomly generate the input with the same shape as the datasets, which allows computer architecture researchers to skip the tedious work of downloading and storing data and more easily analyze the system and architecture characteristics of multi-modal applications. Besides, popular accelerator simulation frameworks such as timeloop [33] simply take the data shape and network shape as input and outputs the latency and energy consumption. MMBench is able to directly provide this abstraction and free users of manual conversion in the simulation.

3.2. Applications in MMBench

MMBench includes nine applications from the five most important multi-modal research domains [24]. For the majority the applications, MMBench provides multiple fusion options covering popular fusion operators [50, 51]. MMBench implements all the applications in SOTA methods to ensure the practical value. The detail of these applications are presented in Table 3.

Multimedia Application: With the development of the internet, multimedia data (language, image, video, and audio) is becoming the largest source of the big data. MMBench rebuilds two of the most representative multimedia applications: (1) *AV-mnist* [34] is assembled from images of handwritten digits and audio samples of spoken digits. (2) *MM-imdb* [32] uses movie titles, metadata, and movie posters to perform multi-label classification of movie genres. We rebuild these applications based on the implementation of *MultiBench*. To make these applications end-to-end and represent the state-of-the-art performance, we replace the fragmented image processing pipeline with an end-to-end *VGG* network [38], and use the pre-trained *ALBERT* model [21] to extract text features.

Affective Computing: Affect computing is the field that studies the perception of human affective states (emotions, sentiment, and personalities) from our natural display of multi-modal signals [24] including spanning language (spoken words), visual (facial expressions, gestures), and acoustic (prosody, speech tone) [24]. MMBench selects two of the most representative affective computing dataset that involving language, video and audio. (1) *MUStARD* is a video corpus used in sarcasm discovery [9]. (2) *CMU-mosei* is the largest dataset of sentence-level sentiment analysis and emotion recognition in real-world online videos [42].

```
if args.model_name=="avmnist_simple_late_fusion":
        channels = 6
        encoders = [LeNet(1, channels, 3).cuda(), LeNet(1, channels, 5).cuda()]
        head = MLP(channels*40, 100, 10).cuda()
        fusion = Concat().cuda()
        filename="./models_save/best_avmnist_simple_late_fusion_-1.pth"
model = MMDL(encoders, fusion, head, has_padding=False).to(device)
model.load_state_dict(torch.load(filename))
train(encoders, fusion, head, traindata, validdata, 30,
      optimtype=torch.optim.SGD, lr=0.05, weight_decay=0.0002, save=filename)
test(model, testdata, no_robust=True)
```

Figure 2: The code snippet of a standard multi-modal application implementation in MMBench.

We rebuild these applications from the original data to make it an end-to-end application. We use *MMSA-FET* [27] to extract features, and including all the modules in the forward pass of the data.

Intelligent Medicine: Modern medical decision-making often involves integrating complementary information from several sources such as lab tests, imaging reports, and patient-doctor conversations. Multi-modal DNNs can help doctors make sense of high-dimensional data and assist them in the diagnosis process. We build this workload based on *ViLMedic* [11], a vision-and-language medical library. We also consider a multi-modal segmentation task that can accurately segment brain tumor from Magnetic Resonance Imaging (MRI) [56]. We rebuild these applications to make them easy to profile with the standard profiling tools.

Smart Robotics: Robotics is also a very important example of multi-modal computing. In order to achieve accurate control of robots, we add many sensors to them and collect multi-modal information (e.g., visual, force etc.). The decision making based on this multi-modal information requires the use of multi-modal networks. MMBench includes two representative robotic tasks: (1) *Mujoco Push* [22], the goal of which is to predict the pose of the object being pushed by the robot end-effector using the collected multi-modal information, i.e., visual (RGB and depth), force, and proprioception sensors. (2) *Vision & Touch* [23], which aims to predict action-conditional learning objectives that capture forward dynamics of the different modalities. We rebuild these workloads to add hooks to profile the different part of the neural network.

Automatic Driving: Automatic Driving normally refers to self-driving vehicles that move without the intervention of a human driver. An autonomous driving systems typically come equipped with both cameras and LiDAR sensors. In MMBench , we modify *TransFuser* [35] which is an architecture for end-to-end driving with two main components: (1) a Multi-Modal Fusion Transformer for integrating information from multiple modalities (single-view image and LiDAR), and (2) an auto-regressive waypoint prediction network. To ease the usage, we extract the TransFuser network and free its dependency on the CARLA simulator [12].

3.3. Implementation Details

The entire MMBench is implemented in PyTorch, while different applications may possess their own dependencies. MMBench provide rich interfaces to enable users to control the workloads. Figure 3 presents a standard MMBench

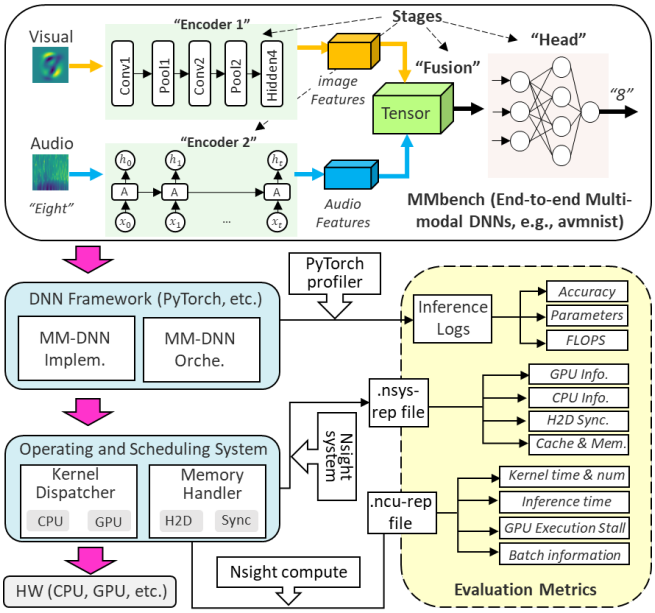


Figure 3: Profiling pipeline in MMBench.

implementation. For a multi-modal application in MMBench, it applies specific encoder networks as the only options. However, it generally includes several different implementations of fusion and head networks. Users can simply include the model name as a command line parameter to choose target multi-modal implementation. Besides, MMBench abstracts the training and inference process and integrates them with profiling tools. Users only need to choose proper options to generate desired metrics.

MMBench targets both servers and edge devices. For servers, the training process and inference process can be done within a single python file. For edge devices, only inference is supported due to limited energy and resources. Models must first be trained on servers. Besides, edge devices such as NVIDIA Jetson series [28] generally adopts a unified memory architecture where GPUs and CPUs share the same physical memory. In this case, adjusting batch size will not affect the memory usage. For large datasets, MMBench will manually split the datasets and inference on partial datasets to ensure the performability.

3.4. Profiling Pipeline

In addition to the representative workloads, MMBench also provides a series of profiling tools based on the most commonly used hardware available today (CPU and NVIDIA GPU) to help locate the system drawbacks and make corresponding improvements. The overall profiling architecture is shown in Figure 3. MMBench provides different command line flags to support different measurements options. To ensure the authority of the measured results, MMBench measures the performance of the network based on standard tools such as, *Python Memory Profiler* [1], *Pytorch Profiler* [2], *NVIDIA Nsight Compute* [30] and *NVIDIA Nsight System* [31].

To ease the usage, MMBench also automates the profiling process using python and shell scripts.

The evaluation metrics can be categorized into three main classes based on the profiling tools and granularity. The first category includes the inference logs directly generated from the applications. Taking advantage of python modules, MMBench is able to provide basic algorithm level information such as model accuracy, parameter number and FLOPs. The second category includes the entire system information such as GPU information, CPU information and the data transfer between host and device. The third category includes more fine-grained GPU information such as kernel information and GPU execution stall reasons since GPU is in charge of nearly all the computation.

4. Evaluation

In this section, we present a detailed evaluation of the proposed MMBench to conduct an in-depth analysis of multi-modal DNNs. We first introduce the experimental platforms and prove the effectiveness of our selected applications. We then investigate the characteristics of multi-modal DNNs from three main aspects: multi-stage execution, workload heterogeneity and execution synchronization.

4.1. Experimental Setups

While MMBench supports various platforms with CPU and NVIDIA GPU, we conduct the following experiments on a GPU server and two edge devices to demonstrate the utility of our benchmark. The GPU server is equipped with two 2.4 GHz Intel 20-core, Xeon 6148 CPUs and four Nvidia RTX 2080Ti GPUs connected via PCI-e ×16 interface with 11 GB of GDDR6 memory. We use a Jetson Nano with 128-core Maxwell, 4GB LPDDR4 and a Jetson Orin with 2048-core Ampere, 32GB LPDDR5 as our edge devices.

4.2. Network-level Characterization

In this section, we characterize the algorithm characteristics of multi-modal DNNs. We analyze their overall performance, and the effect of different fusion schemes and modalities in different applications.

4.2.1. Performance analysis. We first validate that multi-modal DNNs are able to outperform uni-mdoal DNNs. The performance results are presented in different metrics such as accuracy, F-score and MSE. To ensure the practical value of MMBench, we first need to guarantee that all the applications are representative and with high performance. Figure 4 presents the performance of all the applications included in MMBench. For applications with multiple fusion implementations, we only present several results. For *Transfuser*, multiple metrics such as driving score and route completion are used to evaluate its performance. And the *lidar* modality is seldom executed without *image* modality. With specific adjustment and optimization, some of the performance can be further improved.

Observations: *Multi-modal DNNs are proved to outperform uni-modal DNNs in different scenarios. However, multi-modal DNNs generally have various implementations yielding different results. They should be well studied to fully grasp their performance advantage.*

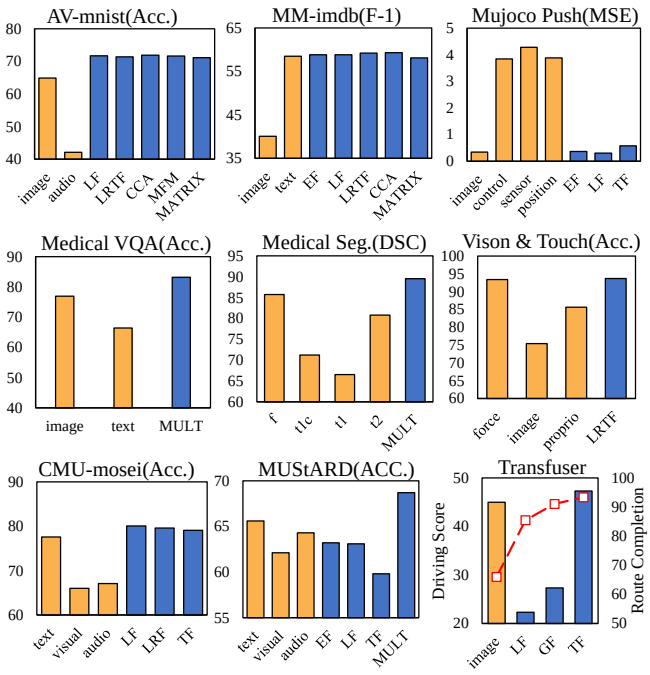


Figure 4: Performance of the applications in MMBench. Lowercase in yellow such as *image* and *audio* indicates uni-modal implementations, upper case in blue such as *EF* and *LF* indicates multi-modal implementations.

4.2.2. Fusion analysis. Most of the current multi-modal researches employ pretrained DNN backbone models as modality encoders and focus on finding more effective fusion or representation methods of different modalities. We examine the influence of different fusion methods on the application performance with same encoders.

Figure 4 also presents different fusion implementations for the datasets. Take *Mujoco Push* as example, the MSE of its implementation in late fusion utilizing *LSTM* is less than 0.3 while the MSE of its implementation in *tensor fusion* reaches 0.58. Similarly, in *MM-imdb*, the maximum performance difference between different fusion schemes can be as large as 1.1 in Micro F1. Some ineffective fusion schemes even lead to lower performance compared with only leveraging single modality. The choice of fusion schemes can lead to significant performance variance.

Observations: *While not significantly influencing the amount of computation for most of the scenarios, different fusion schemes can lead to several percents of absolute performance variance. It's of great importance to design or search for the most effective fusion method.*

4.2.3. Modality analysis. In many real-world scenarios, the importance of different modalities differs depending on the tasks and it is feasible to skip or discard some modality features. Typically, some modalities provide higher accuracy with less computational effort than others in different

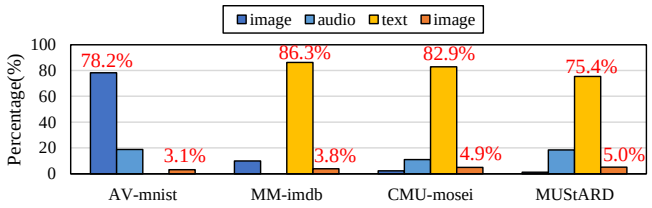


Figure 5: Distribution of mutually exclusive data sample sets correctly processed by different modalities.

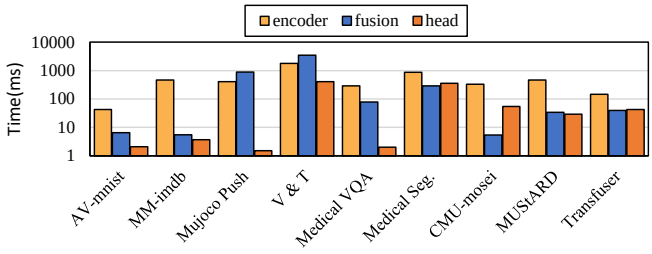


Figure 6: Execution time of a batch of data of the three stages for different MMBench applications.

applications. For example, it has been proven that text-based features perform better than visual or auditory modalities in multi-modal language-emotion analysis tasks [4].

We present the distribution of mutually exclusive data sample sets correctly processed by different modalities in Figure 5. For the four selected datasets, more than 75% of the correct samples can be processed using only a major modality while the major modality differs on different tasks. Only less than 5% of all the correct samples are required to be processed by the multi-modal fusion methods. Therefore, one can only rely on some of the encoders given certain tasks to reduce model complexity. Under such circumstances, intuitively we can simply throttle sensors for less crucial modalities to save energy. However, applying this conventional wisdom is ineffective since it can lead to avoidable task failures resulting from the loss of situation awareness. There exists no retrieval for the extreme conditions where failures occur.

Observations: *Different modalities possess different level of importance in multi-modal DNNs. Smartly activating one of the encoders can fulfill the requirements in most of the cases. There exists room for adaptive execution strategies to achieve a better performance-complexity tradeoff according to the application-specific characteristics.*

4.3. System-/Architecture-level Analysis

In this section, we analyze the system and architecture characteristics of multi-modal DNNs. We analyze them according to their three-stage execution pattern, intra-network heterogeneity and frequent synchronization.

4.3.1. Stage Analysis. As introduced in Section 2, most multi-modal DNNs can be divided into encoder, fusion and head stages. In this section, we analyze the three-stage execution pattern of multi-modal DNNs. We perform the stage analysis by modifying the forward function. We first record the time consumption of the three stages of the datasets, and investigate the resource usage pattern of different stages. Figure 6 presents the execution time of the applications. The execution time distribution depends on specific encoder, fusion and head DNN structures. Generally, encoder stage takes much longer time compared with fusion and head stages. This is because the fusion network takes the learned feature as input, thus having much smaller data size to deal with. However, for complex fusion schemes such as *transformer fusion* in the case of *Mujoco Push* and *Vision & Touch*, it can take even longer time compared with the encoder stage.

We then analyze the resource usage pattern of the MMBench applications in different stages. We trace 5 micro-architectural metrics with *nsight compute* [29], including *DRAM utilization (1)*, *achieved occupancy (2)*, *ipc (3)*, *gld efficiency* (4) and *gst efficiency (5)*. The detailed results are presented in Figure 7. Generally, the encoder stages present higher *DRAM utilization*, *IPC* and *GPU occupancy* compared with fusion and head stages since they include more computation. For *gld efficiency* and *gst efficiency*, all the stages presents nearly the same resource usage pattern. For complex fusion schemes such as *transformer fusion* in *Mujoco Push*, although it takes nearly 3× more execution time compared with the encoder stage, it does not consume much more resources. While it shows slight increase in *IPC* and *GPU occupancy*, the *DRAM utilization* of the encoder stage is still higher.

Observations: *There exists significant time and resource imbalance in different stages, which leads to possible resource under-utilization. If we assign fixed resources to a multi-DNN application according to its encoder stage, more than half of the resources, especially memory, may actually stay idle when the application enters the fusion and head stages.*

4.3.2. Heterogeneity Analysis. Each of the multi-modal encoder sub-network and the fusion network of a multi-modal DNN approximates an independent uni-modal network. Therefore, there is a high degree of heterogeneity within the multi-modal DNNs. In this section, we first analyze the GPU kernel type breakdown of multi-modal DNNs. We then delve further to analyze the kernel-level information of some hotspot kernels. We only select one of the implementations for all the applications.

Figure 8 presents the GPU kernel type breakdown for the applications in MMBench. We classify all the GPU operations into 8 categories including convolutions (*Conv*), batch normalization (*BNorm*), element-wise operation (*Elewise*), pooling (*Pooling*), relu activation (*Relu*), general matrix multiply (*Gemm*), reduce (*Reduce*) and else (*Other*). In this regard, each kernel type contains a subset of function calls that execute similar tasks. We observe that different stages within a same application are dominated by different type of operations, not to mention the difference between different applications. Besides, the encoder networks for different modalities are highly diverse. Some applications, such as *AV-mnist*, apply same encoder network (Lenet) for both

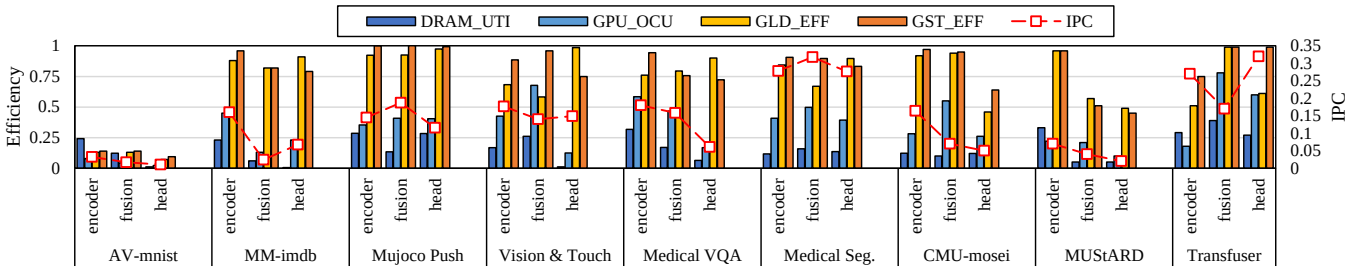


Figure 7: Resource usage of the three stages for different MMBench applications.

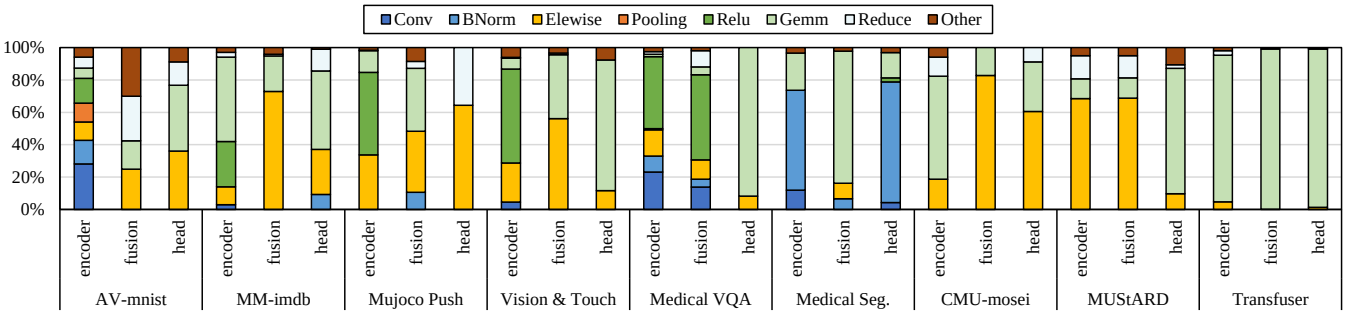
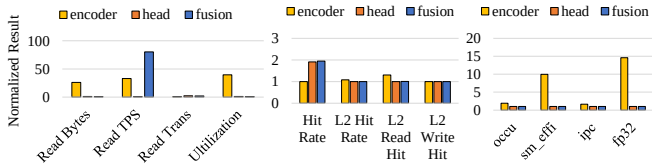
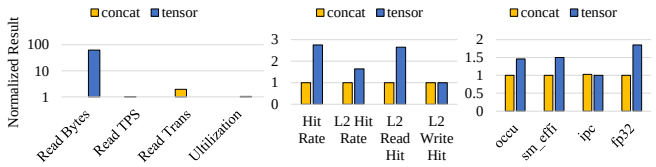


Figure 8: Kernel operation breakdown of the three stages for different MMBench applications.



(a) Hotspot kernel: *Reduce* in different stages



(b) Hotspot kernel: *Element-wise* in different fusion methods

Figure 9: Dedicated kernel comparison different stages and fusion methods on *AV-mnist*. The result is normalized.

modalities. However, *MM-imdb* apply *VGG* and *Albert* to encoder the different modalities. While *VGG* is dominated by *Gemm* (72%), *Albert* is dominated by *relu* (66%). Different acceleration strategies are required for these two encoders.

We further choose two hotspot kernels in the case of *AV-mnist* and analyze their fine-grained performance in Figure 9. We study the computation, cache and memory patterns of two specific GPU kernels in different stages and different fusion implementations. The resource usage of the same kernel in different fusion methods is basically at the same level despite a significant increase in DRAM read bytes. However, when it comes to the same kernel in different stages, its average resource usage can vary from 15× in the total number of *fp32* operations to 80× in *read TPS*. The large difference in memory and compute resources possibly results from the input data size, since fusion and head only handle the learned representations from the encoder stage.

Observations: *There exists different dominant operations in different subnets, and the same operations may perform differently in different stages. In this regard, it is hard to find a universal optimization for the whole multi-modal application. Multi-modal applications must be analyzed first to identify the bottlenecks. It is hard to design specialized hardware accelerators for multi-modal DNN applications.*

4.3.3. Synchronization Analysis. In this section, we analyze the synchronization problem of multi-modal DNNs. From application level, there exists the problem of modality synchronization. The fusion stage must wait until the completion of all modalities. From network level, multi-modal DNNs suffer from data synchronization. There exists additional intermediate data and data preparation operations which can even overweight GPU computation. We first record the execution time of different modalities and then investigate the proportion of CPU+Runtime/ GPU execution.

Modality synchronization. We record the execution time for different modalities. In Figure 10, it's obvious that the execution time of different modalities are different. This problem is especially usual for multi-modal tasks involving image modality since image modality generally produces larger amounts of data and require more computation. For example, the straggler (*uni2: image*) modality in *Mujoco Push* takes up to 4.09× of inference time compared to other modalities. If executed concurrently, nearly 75% of the resources assigned to the application will stay idle for more 77% of the entire encoder execution.

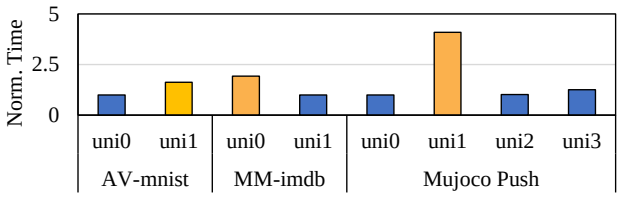


Figure 10: Execution time for different modalities for MMBench applications.

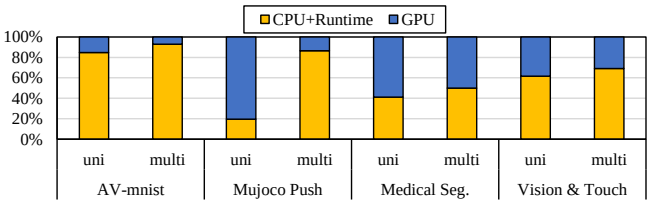


Figure 11: Comparison between synchronization and computation for MMBench applications.

Data synchronization. Most of the network computation are executed on GPUs, and CPUs are mainly in charge of data processing operations, such as *to* and *copy*. Thus we consider that a higher *CPU+Runtime* ratio indicates more data synchronization operations as GPU are more frequently kept stalled for lack of data. We choose several applications in different research domains to investigate their inference time breakdown. The detailed results are shown in Figure 11. We can observe that for all applications, the multi-modal implementation possess larger proportion of *CPU+Runtime* operations compared with the uni-modal implementations. Complex fusion such as *Mujoco push* can lead to a significant increase in *CPU+Runtime* of 66%.

Observations: *Multi-modal DNNs suffer from two-level of synchronization. From application level, its encoder subnets requires modality synchronization before the fusion stage. The fusion network must always wait for the straggler. From operator level, lengthy intermediate data operations lead to frequent data synchronization. These altogether leads to the resource under-utilization problem, as GPUs may stay idle for most of the application time.*

5. Two Case Studies Using MMBench

5.1. Effect of Batch Size

Beyond simply executing the multi-modal DNN applications, MMBench also provides multiple tuning knobs to study the effect of different parameters and help adjust the system. Here we provide a case study, demonstrating how MMbench help explore the effect of batch size on multi-modal DNNs compared with uni-modal DNNs.

Generally, when a batch of tasks arrive, the operating system schedules the appropriate kernels to handle those tasks. If we ignore the computational differences among various kernels of different sizes, a batch size of 400 tasks will be executed in 10× less time than a batch of 40 tasks. However, this is impossible during real execution

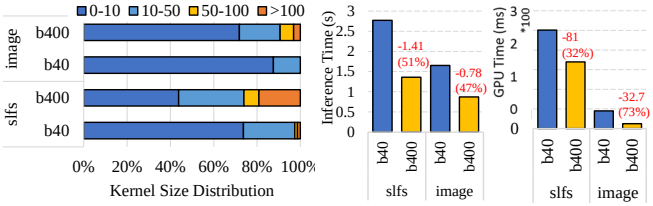
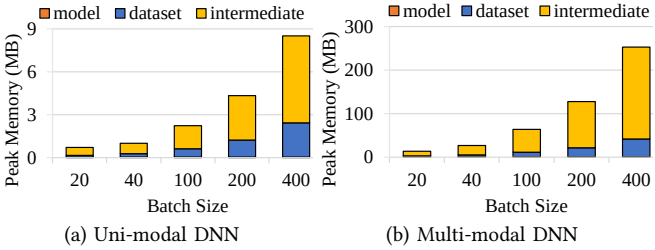


Figure 12: Larger batch size can accelerate the execution of multi-modal DNNs on *AV-mnist*. *Slfs* is a multi-modal implementation, and *image* is the uni-modal counterpart. *b* refers to batch size.



(a) Uni-modal DNN (b) Multi-modal DNN

Figure 13: Peak memory for processing models, datasets, and intermediate results on *AV-mnist*.

due to resource contention and constraints. The current OS often leverages larger kernels which yield better tradeoff between GPU time and non-GPU time (e.g., data transfer time, synchronization time etc.) to process a large batch of tasks. In this regard, it is more beneficial to process multi-modal DNN tasks in large batch size. Figure 12 shows our analysis. We consider 10000 inference tasks which are scheduled with batch size of 40 and 400 respectively.

We first analyze the distribution of kernels of different sizes. Based on the GPU execution time of each kernel, we divide the kernels into four different kernel sizes. Figure 12 illustrates the comparison results of uni-modal and multi-modal DNNs. *0-10* indicates a small kernel, where the kernel executes in less than 10 microseconds. *>100* indicates a large kernel, where the kernel executes in more than 100 microseconds. The leftmost result shows that existing operating system (OS) uses more large kernels whose execution time exceed 50 microseconds to process a larger batch size with 400 tasks. Meanwhile, the OS calls more large kernels to process the multi-modal DNN tasks. The results on the right show that a 10x increase in batch task size does not reduce the processing latency by 10x.

Besides, as shown in Figure 13, larger batch size leads to higher peak memory usage for model, dataset and intermediate features. The model sizes remain generally the same, while the dataset and intermediate features present a linear order relative to batch size. Multi-modal DNNs also tend to produce higher proportion of intermediate data. When changing batch size, multi-modal DNNs are easier to achieve GPU memory capacity since they involve

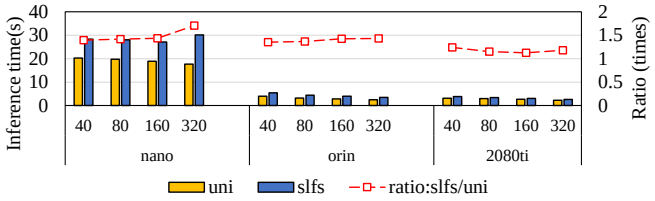


Figure 14: Inference time of *AV-mnist* on GPU server and edge devices with the change of batch size. *slfs* refers to an implementation of multi-modal with 31x parameters.

more intermediate features with multiple modalities and additional fusion networks.

Observations: *Our analysis shows that different components of multi-modal DNNs benefits differently from batch size. The GPU time of multi-modal DNNs decrease in a smaller scope, which possibly results from the kernel composition of the networks. Besides, batch size increase leads to higher growth rate in peak memory usage for multi-modal DNNs.*

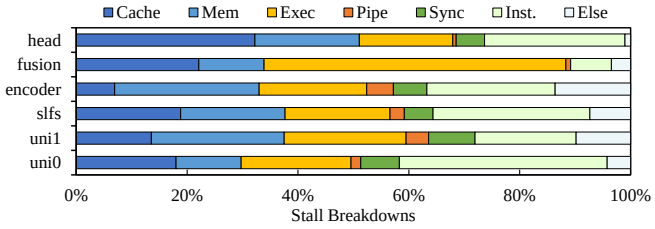
5.2. Migration to Edge Computing

In recent years, there has been an increasing trend to deploy DNN models at the edge due to the connectivity, latency and privacy concerns of transferring data to the cloud. Therefore, we also characterize the features of multi-modal DNNs at the edge. We run *AV-mnist* on one of the most representative AIoT boards, i.e., Jetson Nano.

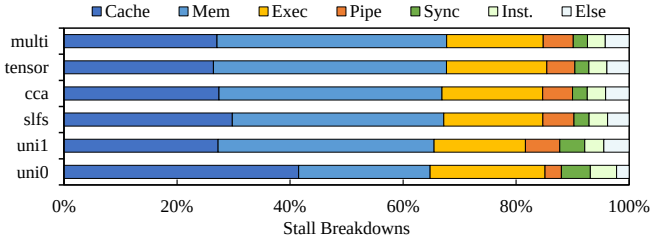
Figure 14 presents the inference time of *AV-mnist* on both GPU servers and edge devices. On Jetson nano where resources are limited, 6.48× more time is needed compared with GPU servers. With the increase of batch size, while the latency of GPU server is constantly decreasing, the latency of Jetson nano is even higher when batch size reaches 320. It is because certain resources are used up. On Jetson orin with abundant resources, the multi-modal DNNs perform similarly as on GPU servers. The ratio of multi-modal execution time compared with uni-modal is higher on Jetson nano and orin, since GPU servers possess more idle resources.

In Figure 15-(a) and (b), we illustrate the execution stall breakdown and resource usage patterns of multi-modal DNN both on edge devices and on GPU servers. We divide the stall reasons into 7 main categories: *cache dependency (Cache), memory dependency (Mem), execution dependency (Exec), busy pipeline (Pipe), synchronization blocked (Sysn), instruction not fetched (Inst.), other stalls (Else).* The stall caused by *execution dependency* and *instruction not fetch* increases dramatically on edge devices, while *memory dependency* and *cache miss* are the main causes of stall on GPU servers. It possibly results from the lack of computing power so that requisite operations cannot be finished in time. As shown in Figure 15-(c), on edge devices with limited resources, *DRAM utilization* is almost always kept at the highest level. Unlike GPU servers in Figure 7, fusion stage now possesses higher *GPU occupancy* on edge devices.

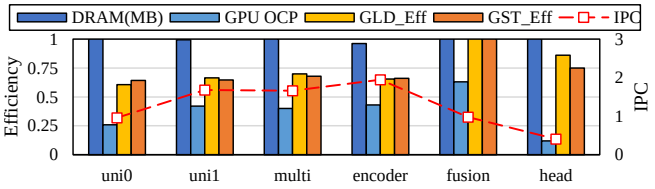
Observations: *Migration to edge devices leads to higher latency and new bottlenecks. Due to limited power and resources,*



(a) Breakdown of stall cycles on Jetson Nano



(b) Breakdown of stall cycles on 2080ti



(c) Computation and memory usage on Jetson Nano

Figure 15: Execution stall breakdown and resource usage on edge devices. *uni0* refers to audio, *uni1* refers to image.

the inference time grows dramatically when we switch from uni-modal DNN to multi-modal DNNs even on small datasets. It would be a huge challenge to enable multi-modal DNNs on edge devices. Some of the modalities may be skipped to guarantee the QoS on edge devices as long as the result meets the requirement.

6. Conclusion

We present MMBench, an open-source benchmark suite for end-to-end cloud and IoT multi-modal neural networks. The suite includes multiple representative multi-modal computing applications, such as multimedia analysis, affective computing, medical analysis, etc. We use MMBench to study the system and architectural implications of multi-modal neural networks across different computing stacks and conclude three unique characteristics. We also provide two case studies to demonstrate how MMBench guides the system and architecture designs. We expect that our work could pave the way for better system and architecture research for multi-modal computing.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.62122053). Corresponding authors are Xiaofeng Hou and Chao Li. We thank all the anonymous reviewers for their valuable comments and suggestions.

Appendix

1. Abstract

This artifact reproduces the results shown in Evaluation part, showing how we use MMBench to conduct an in-depth analysis of multi-modal DNNs.

2. Artifact check-list (meta-information)

- **Program:** Pytorch Profiler NVIDIA Nsight Systems, and NVIDIA Nsight Compute
- **Compilation:** PyTorch
- **Model:** multi-modal models made up of Lenet, VGG, Transformer, Albert, Resnet
- **Data set:** Avmnist, MMimdb, CMU-MOSEI, Sarcasm, Medical VQA, Medical Segmentation, MuJoCo Push, Vison & Touch, TransFuser
- **Run-time environment:** Linux5.4.0-113.x86_64
- **Hardware:** GPU: NVIDIA RTX 2080Ti; edge device: Jetson Nano, Jetson Orin
- **Run-time state:** Run-time varies based on model type and configuration
- **Execution:** Executing commands can be found on GitHub
- **Output:** PyTorch Profiler, NVIDIA Nsight Systems and NVIDIA NSIGHT Compute trace files
- **Experiments:** Analyzing the profiling results of the inference phase of each model and investigating GPU behavior of the models
- **How much disk space required (approximately)?:** 100 GB.
- **How much time is needed to prepare workflow (approximately)?:** About one hour to install related python packages and install NVIDIA Nsight Systems and NVIDIA Nsight Compute tools.
- **How much time is needed to complete experiments (approximately)?:** 4 hours.
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** CCA 4.0 International
- **Archived (provide DOI)?:** 10.5281/zenodo.8266664

3. Description

3.1. How to access. Our source codes are available at github (https://github.com/xfhelen/MMBench) and zenodo (https://zenodo.org/record/8266664)

3.2. Hardware dependencies. Most of the benchmarks consume a few megabytes when running. so anything modern should be fine. However, The GPU server We use is equipped with two 2.4 GHz Intel 20-core, Xeon 6148 CPUs and four Nvidia RTX 2080Ti GPUs connected via PCI-e ×16 interface with 11 GB of GDDR6 memory. We use a Jetson Nano with 128-core Maxwell, 4GB LPDDR4 and a Jetson Orin with 2048-core Ampere, 32GB LPDDR5 as our edge devices.

3.3. Software dependencies. The operation system we used is Linux5.4.0-113.x86_64. The software framework is Pytorch.

3.4. Data sets. Avmnist, MMimdb, CMU-MOSEI, Sarcasm, MuJoCo Push, Vison & Touch, Medical VQA, Medical Segmentation, TransFuser

3.5. Models. Multi-modal models made up of Lenet, VGG, Transformer, Albert, Resnet, ...

4. Installation

About python environment, you just need to git clone our repository and use conda to create a new environment by running command:

```
conda env create -f environment.yml
-n new_env_name
```

For more information related to PyTorch Profiler, please refer to the tutorial (https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html).

For NVIDIA Nsight Systems and NVIDIA Nsight Compute, download them from the following links: https://developer.nvidia.com/gameworksdownload#?dn=nsight-systems-2022-3 and run the installer.

5. Experiment workflow

Before running the benchmark, you should download the datasets following the instruction of `README` file of the application that you want to run.

Then make sure the working directory is the `MMBench` folder. The scripts to run are stored in the `scripts` folder, and all the applications are stored in the `applications` folder. You can run the python code of a application directly to test whether it can work on you python environment, for example:

```
python applications/CMU-MOSEI/
affect_late_fusion.py
```

You can also skip the aforementioned process to run the benchmark directly. If you want to obtain data such as DRAM utilization, achieved occupation, IPC, GLD efficiency, and GST efficiency, you can run the script `ncu_metric.sh`. Here is an example of how to run it:

```
./scripts/ncu_metric.sh applications/CMU-MOSEI
/affect_late_fusion.py applications/
CMU-MOSEI/ncu_info.csv
```

The last parameter in the example above is used to specify the location where the measurement data will be stored. In addition, our benchmark also provides the functionality to measure specific stages of the encoder, fusion, or head. You can specify a particular stage for parameter measurement.

```
./scripts/ncu_metric.sh applications/CMU-MOSEI
/affect_late_fusion.py applications/CMU-MOSEI
/ncu_info_encoder.csv --option encoder
```

If you want to obtain the allocation of GPU operators during the code execution, you can run the `nsys_metric.sh` script. Here is an example:

```
./scripts/nsys_metric.sh applications/CMU-MOSEI
/affect_late_fusion.py
```

The results obtained will be ultimately stored in the file `scripts nsys_temp_file.txt`.

6. Evaluation and expected results

For NVIDIA NSIGHT Compute, you can get profiling results as Figure 16 shows:

```
Output file path: applications/Vison&Touch/ncu_info.csv
Metric: DRAM utilization, Average Value: 17.131235955056173
Metric: achieved occupancy, Average Value: 37.859348314606784
Metric: IPC, Average Value: 0.15757303370786654
Metric: GLD efficiency (global load efficiency), Average Value: 68.62361797752806
Metric: GST efficiency (global store efficiency), Average Value: 89.53728089887643
```

Figure 16: Results of NSIGHT Compute

For Pytorch Profiler, you can get profiling results as Figure 17 shows:

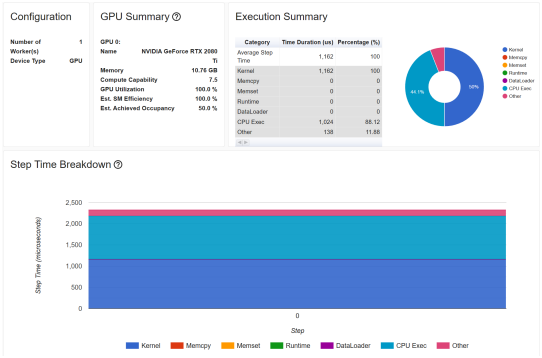


Figure 17: Results of Pytorch Profiler

The obtained results should reflect the figures in the article with python-based collection and processing.

7. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

References

[1] "Python memory profiler," https://pypi.org/project/memory-profiler/.

[2] "Pytorch profiler," https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html/.

[3] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. M. Brooks, "Fathom: reference workloads for modern deep learning methods," in *2016 IEEE International Symposium on Workload Characterization, IISWC 2016*.

[4] M. S. Akhtar, D. S. Chauhan, D. Ghosal, S. Poria, A. Ekbal, and P. Bhattacharyya, "Multi-task learning for multi-modal emotion recognition and sentiment analysis," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*.

[5] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "Embench: Quantifying performance variations of deep neural networks across modern commodity devices," *CoRR*, vol. abs/1905.07346, 2019.

[6] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the SMO algorithm," in *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004)*.

[7] T. Baltrusaitis, C. Ahuja, and L. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[8] T. Baruah, K. Shivdikar *et al.*, "Gnnmark: A benchmark suite to characterize graph neural network training on gpus," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021*.

[9] S. Castro, D. Hazarika, V. Pérez-Rosas, R. Zimmermann, R. Mihalcea, and S. Poria, "Towards multimodal sarcasm detection (an _obviously_ perfect paper)," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019*.

[10] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, 2017.

[11] J.-B. Delbrouck, K. Saab, M. Varma, S. Eyuboglu, P. Chambon, J. Dunnmon, J. Zambrano, A. Chaudhari, and C. Langlotz, "Vilmedic: a framework for research at the intersection of vision and language in medical ai," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL, 2022*.

[12] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," in *1st Annual Conference on Robot Learning, CoRL 2017*.

[13] U. Gupta, C. Wu *et al.*, "The architectural implications of facebook's dnn-based personalized recommendation," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 20200*.

[14] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *IEEE International Symposium on Workload Characterization, IISWC 2019*.

[15] X. Hou, C. Li, J. Liu, L. Zhang, Y. Hu, and M. Guo, "Ant-man: towards agile power management in the microservice era," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, 2020.

[16] X. Hou, M. Liang, C. Li, W. Zheng, Q. Chen, and M. Guo, "When power oversubscription meets traffic flood attack: Re-thinking data center peak load management," in *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, August 05-08, 2019*, 2019.

[17] X. Hou, J. Liu, C. Li, and M. Guo, "Unleashing the scalability potential of power-constrained data center in the microservice era," in *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, August 05-08, 2019*, 2019.

[18] X. Hou, J. Liu, X. Tang, C. Li, J. Chen, L. Liang, K. Cheng, and M. Guo, "Architecting efficient multi-modal aiot systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023, Orlando, FL, USA, June 17-21, 2023*, 2023.

[19] X. Hou, C. Xu, J. Liu, X. Tang, L. Sun, C. Li, and K.-T. Cheng, "Characterizing and understanding end-to-end multi-modal neural networks on gpus," *IEEE Computer Architecture Letters (CAL)*, 2022.

[20] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2021*.

[21] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," in *8th International Conference on Learning Representations, ICLR 2020*.

[22] M. A. Lee, B. Yi, R. Martín-Martín, S. Savarese, and J. Bohg, "Multimodal sensor fusion with differentiable filters," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*, 2020.

[23] M. A. Lee, Y. Zhu, P. Zachares, M. Tan, K. Srinivasan, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Learning multimodal representations for contact-rich tasks," *IEEE Trans. Robotics*, 2020.

[24] P. P. Liang, Y. Lyu, X. Fan, Z. Wu, Y. Cheng, J. Wu, L. Chen, P. Wu, M. A. Lee, Y. Zhu, R. Salakhutdinov, and L. Morency, "Multibench: Multiscale benchmarks for multimodal representation learning," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021*.

[25] S. Liu, B. Yu, J. Tang, Y. Zhu, and X. Liu, "Communication challenges in infrastructure-vehicle cooperative autonomous driving: A field deployment perspective," 2022.

[26] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, and J. Dai, "Comparison and benchmarking of AI models and frameworks on mobile devices," *CoRR*, vol. abs/2005.05085, 2020.

[27] H. Mao, Z. Yuan, H. Xu, W. Yu, Y. Liu, and K. Gao, "M-SENA: an integrated platform for multimodal sentiment analysis," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*.

[28] NVIDIA, "Nvidia jetson nano developer kit," 2020.

[29] ——, "Cuda toolkit v11.7.0," https://docs.nvidia.com/cuda/profiler-users-guide/index.html, 2022.

[30] ——, "Nsight compute," https://developer.nvidia.com/nsight-compute, 2022.

[31] ——, "Nsight systems," https://developer.nvidia.com/nsight-systems, 2022.

[32] J. E. A. Ovalle, T. Solorio, M. Montes-y-Gómez, and F. A. González, "Gated multimodal units for information fusion," in *5th International Conference on Learning Representations, ICLR 2017*.

[33] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. S. Emer, "Timeloop: A systematic approach to DNN accelerator evaluation," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2019*.

[34] H. Pham, P. P. Liang, T. Manzini, L. Morency, and B. Póczos, "Found in translation: Learning robust joint representations by cyclic translations between modalities," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*.

[35] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021*.

[36] V. J. Reddi, C. Cheng *et al.*, "Mlperf inference benchmark," in *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020*.

[37] T. Schefke, "Deepbench: Open-source tools for a.i. in the sky," 2020.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*.

[39] Z. Sun, P. K. Sarma, W. A. Sethares, and Y. Liang, "Learning relationships between text, audio, and video via deep canonical correlation for multimodal language analysis," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*.

[40] F. Tang, W. Gao *et al.*, "Aibench: An industry standard AI benchmark suite from internet services," *CoRR*, 2020.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, 2017*.

[42] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[43] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?" in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*.

[44] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019*.

[45] Y. Wu, E. Y. Chang, K. C. Chang, and J. R. Smith, "Optimal multimodal fusion for multimedia data analysis," in *Proceedings of the 12th ACM International Conference on Multimedia, 2004*.

[46] A. Wudenhe and H. Tseng, "Tpupoint: Automatic characterization of hardware-accelerated machine-learning behavior for cloud computing," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021*.

[47] S. L. Xi, Y. Yao, K. Bhardwaj, P. N. Whatmough, G. Wei, and D. Brooks, "SMAUG: end-to-end full-stack simulation infrastructure for deep learning workloads," *ACM Trans. Archit. Code Optim.*, 2020.

[48] X. Xie, X. Hu, P. Gu, S. Li, Y. Ji, and Y. Xie, "Nnbench-x: Benchmarking and understanding neural network workloads for accelerator designs," *IEEE Comput. Archit. Lett.*, 2019.

[49] P. Xu, X. Zhu, and D. A. Clifton, "Multimodal learning with transformers: A survey," *CoRR*, 2022.

[50] Z. Xu, D. R. So, and A. M. Dai, "Mufasa: Multimodal fusion architecture search for electronic health records," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*.

[51] Y. Yin, S. Huang, and X. Zhang, "BM-NAS: bilevel multimodal neural architecture search," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*.

[52] A. Zadeh, M. Chen, S. Poria, E. Cambria, and L. Morency, "Tensor fusion network for multimodal sentiment analysis," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*.

[53] C. Zhang, Z. Yang, X. He, and L. Deng, "Multimodal intelligence: Representation learning, information fusion, and applications," *IEEE J. Sel. Top. Signal Process.*, 2020.

[54] Q. Zhang, L. Zha, J. Lin, D. Tu, M. Li, F. Liang, R. Wu, and X. Lu, "A survey on deep learning benchmarks: Do we still need new ones?" in *Benchmarking, Measuring, and Optimizing - First BenchCouncil International Symposium, Bench 2018*.

[55] Q. Zhang, X. Li, X. Che, X. Ma, A. Zhou, M. Xu, S. Wang, Y. Ma, and X. Liu, "A comprehensive benchmark of deep learning libraries on mobile devices," in *The ACM Web Conference, WWW, 2022*.

[56] Y. Zhang, N. He, J. Yang, Y. Li, D. Wei, Y. Huang, Y. Zhang, Z. He, and Y. Zheng, "mmformer: Multimodal medical transformer for incomplete multimodal learning of brain tumor segmentation," in *Medical Image Computing and Computer Assisted Intervention, MICCAI 2022*.

[57] X. Zhou and B. Bhanu, "Feature fusion of side face and gait for video-based human identification," *Pattern Recognit.*, 2008.